A Graph-Based Agent Approach to Numerical Reasoning Question Answering

Quang Nguyen[1] **Vu Thanh Dat Ha**[1] **Thanh Tuan Le**[1] **Huan Vu**[2]
[1] VNPT AI [2] National Economics University
{quangngm, thanhdathv, lethanhtuan}@vnpt.vn
{huanv}@neu.edu.vn

Abstract

Numerical reasoning in finance requires high accuracy and transparency, yet directly prompting Large Language Models (LLMs) for multi-step calculations is often unreliable. While modern LLMs possess long-context capabilities, complex financial documents still pose significant reasoning challenges. We propose a four-step AI agent pipeline. Its core contribution is a planning stage that utilizes n-sampling to generate multiple reasoning paths, allowing the agent to select the most coherent and accurate solution. The system is designed for the two competition subtasks: Subtask 1 (models $\leq$13B parameters), for which we use Qwen3-8B, and Subtask 2 (any open-source model size), for which we use Qwen3-32B. Our method proved highly effective on the VLSP 2025 Numerical Reasoning QA private test set, achieving the highest Execution Accuracy (84.00%) in Subtask 2 and securing a top-3 rank in Subtask 1 with 79.14% Execution Accuracy, demonstrating the superiority of our structured, multi-path reasoning approach.

1 Introduction

Financial analysis is critical for assessing business performance, but the sheer volume and complexity of financial documents make manual analysis a significant challenge for human experts (Jerven, 2013; MacKenzie, 2008). This has motivated the development of automated systems to perform deep analysis of financial data. However, numerical reasoning in the financial domain poses unique challenges compared to general-domain Question Answering (QA) tasks like DROP (Dheeru Dua and Gardner, 2019). Unlike general questions, financial QA requires systems to perform complex, multi-step numerical reasoning by synthesizing information from heterogeneous sources, including both unstructured text and tables (Chen, 2021). Approaches that rely on directly prompting Large Language Models (LLMs) for these calculations often prove unreliable and lack the necessary transparency for high-stakes financial applications. For a concrete illustration of the task, which requires generating both a reasoning program and a final answer, see Example 1.1.

In this paper, we address these challenges by proposing a structured, inference-only AI agent pipeline. Our approach systematically breaks down the problem into four distinct stages: Question Decomposition, Data Extraction, Planning, and Equation Extraction. The cornerstone of our method is the planning stage, where we employ n-sampling. Instead of committing to a single line of reasoning, this technique generates multiple potential execution plans. This multi-path exploration allows the agent to evaluate and select the most robust and logically sound reasoning program, significantly improving accuracy over single-pass generation methods. The pipeline utilizes the Qwen3-8B model for Subtask 1 and the more powerful Qwen3-32B for Subtask 2. We specifically selected the Qwen3 model family (Qwen Team, 2025) due to its strong reasoning capabilities, which are well-suited for tasks requiring coding, logical calculations, and step-by-step thinking. This architecture makes it easier to trace the model's reasoning process, which is crucial for analysis and future improvements. Our methodology was validated in the VLSP 2025 Numerical Reasoning QA competition, where it achieved outstanding results on the highly competitive private test set. In Subtask 2, our system demonstrated its superiority by achieving the highest Execution Accuracy (84.00%) among all competitors, paired with a Program Accuracy of 74.07%. In Subtask 1, it also delivered a strong performance with 79.14% EA and 69.82% PA, securing top-tier rankings in both categories. The main contributions of this work are threefold:

1. We design a robust, four-step agent pipeline specifically tailored for financial numerical

reasoning.

2. We introduce the novel application of n-sampling for generating and optimizing multiple reasoning plans, enhancing the model's accuracy and robustness.

3. We demonstrate state-of-the-art performance on a competitive benchmark, confirming that structured agentic workflows significantly outperform direct LLM prompting for complex reasoning tasks.

Example 1.1: Numerical Reasoning QA Example

Question: Doanh thu thuần năm 2023 gấp bao nhiêu lần doanh thu thuần năm 2022?
Context: *(Abbreviated; see task description for details containing a table with revenue figures for 2023 and 2022.)*
Program: divide(914, 391)
Answer: 2.337595907928389

2 Related Work

Numerical reasoning in question answering has been explored in various benchmarks. General-domain tasks include DROP (Dheeru Dua and Gardner, 2019), focusing on reading comprehension with discrete reasoning, and MathQA (Aida Amini and Hajishirzi, 2019), targeting mathematical word problems. In the financial domain, FinQA (Chen, 2021) introduced a dataset for numerical reasoning over financial reports, emphasizing interpretable reasoning programs. TAT-QA (Zhu, 2021) extends this to hybrid tabular and textual content in finance, requiring diverse numerical operations such as addition, subtraction, multiplication, and division. ConvFinQA (Chen, 2022) further explores conversational aspects, emphasizing chain-of-numerical reasoning in finance dialogues. Recent works have proposed specialized models and approaches to tackle these challenges. For instance, TAT-LLM (Zhu et al., 2024) is a language model fine-tuned for discrete reasoning over financial tabular and textual data, demonstrating improved performance on benchmarks like TAT-QA. ELASTIC (Zhang and Moshfeghi, 2022) introduces an adaptive symbolic compiler for numerical reasoning, allowing more flexible program generation. Additionally, multi-agent frameworks, such as the one proposed by Lee et al. (2024), incorporate a critic agent to reflect on reasoning steps and improve answer quality in financial QA tasks. Case-based reasoning approaches (Kim, 2024) have also been explored to address multi-step numerical problems by retrieving and adapting similar past cases. More recent benchmarks like FinanceReasoning (Tang et al., 2025) and FinMMR (Tang, 2025) aim to make financial numerical reasoning more credible, comprehensive, and challenging, including multimodal elements and broader financial concepts. The VLSP 2025 Numerical Reasoning QA extends these efforts by focusing on Vietnamese financial reports and evaluating both execution accuracy (numerical correctness) and program accuracy (logical equivalence of reasoning programs). Our graph-based pipeline builds on these foundations, integrating advanced planning with n-sampling and parallel processing to enhance transparency and efficiency in financial QA.

3 Task and Dataset

3.1 Task Definition

The VLSP 2025 Numerical Reasoning QA task challenges systems to reason over financial documents. Formally, given a question q and its corresponding context C, which consists of unstructured text (pre_text, post_text) and a structured table (table), the task is to learn a mapping $f : (q, C) \rightarrow (a, p)$. Here, a is the final numerical answer (*answer*), and p is the transparent, step-by-step reasoning program (*program*) that produces a.

The competition is structured into two distinct subtasks, both sharing the same dataset but with different constraints on model size:

- **Subtask 1** restricts participants to using models with 13 billion parameters or fewer ($\leq$13B).
- **Subtask 2** allows for the use of models of any size, with the exclusion of proprietary, API-only models (e.g., GPT series, Gemini).

3.2 Dataset

The dataset provided, denoted as $\mathcal{D} = \{d_i\}_{i=1}^{N}$, consists of N samples. Each sample d_i is a tuple containing:

- context: The surrounding text and table data from a Vietnamese financial report.
- question: The user's question in Vietnamese.

- program: The gold reasoning program.
- exe_ans: The gold numerical answer derived from executing the program.

The training set is composed of the translated FinQA dataset (Chen, 2021) and a collection of Vietnamese financial reports spanning from 2020 to 2025. The evaluation is conducted on public and answer-hidden private test sets, which only provide the context and question.

4 Proposed Method

We introduce a modular, multi-stage agentic workflow designed to systematically deconstruct and solve financial numerical reasoning problems. Our pipeline models the reasoning process as a directed acyclic graph, ensuring a structured and transparent execution flow. As illustrated in Figure 1, the architecture comprises four distinct modules: a Subquery Generator, a Subquery Answerer, a Plan and Scheduler, and an Equation Extractor.

4.1 Question Decomposition

To mitigate the risk of reasoning errors, a key challenge for modern LLMs, despite their capacity for long context windows, is maintaining focused reasoning amidst noisy data. Even when an entire document is ingested, the model's attention can drift, causing it to struggle with discerning key numerical figures from irrelevant details and failing to grasp the precise relationships between them. This increases the risk of reasoning errors, such as hallucination or overlooking critical values. To counteract this, our pipeline adopts a "divide and conquer" strategy, first decomposing the main question q into atomic, fact-seeking subqueries. This approach explicitly separates the task of identifying the necessary numerical evidence from the subsequent, more complex task of calculation. By forcing the model to focus on one piece of information at a time, we minimize the distracting influence of the broader context and significantly enhance the reliability of the numerical grounding for the final reasoning step. The decomposition is performed by an LLM-based generator function, G_{sq}, which is prompted to identify the core numerical entities required to answer q.

$$SQ = G_{\text{sq}}(q, C) \tag{1}$$

where $SQ = \{sq_1, sq_2, \ldots, sq_k\}$ is the resulting set of k self-contained subqueries that probe for specific values within the context C. G_{sq} is implemented as a prompted LLM call that takes the question and context and is instructed to generate a JSON object of fact-seeking queries.

4.2 Grounded Data Extraction

Each generated subquery $sq_j \in SQ$ is then independently executed against the full context C. This stage grounds the subsequent reasoning process in verifiable numerical evidence by isolating the data extraction task. An LLM-based answering function, A_{sq}, is responsible for this information retrieval. Formally, this function maps each subquery to its corresponding value v_j, yielding a set of numerical arguments V.

$$V = \{v_j \mid v_j = A_{\text{sq}}(sq_j, C), \forall sq_j \in SQ\} \tag{2}$$

The resulting set $V = \{v_1, v_2, \ldots, v_k\}$ provides the necessary numerical and textual arguments for the subsequent program synthesis stage. A_{sq} is implemented as an independent LLM call for each subquery.

4.3 Multi-Path Program Generation

Rather than committing to a single, potentially flawed reasoning path, our method explores a diverse solution space by generating multiple independent output sequences. We utilize the n-sampling parameter to generate a set $\mathcal{P}_{\text{cand}}$ of n distinct candidate reasoning programs (in our experiments, $n = 15$).

$$\mathcal{P}_{\text{cand}} = P_{\text{n-sample}}(V, C, q, \mathcal{T}) \tag{3}$$

This approach allows the agent to consider multiple hypotheses for combining the extracted numerical arguments V using a predefined toolset $\mathcal{T}$ (e.g., add, divide). By exploring a variety of computational graphs, this method significantly enhances robustness against common reasoning fallacies. $P_{\text{n-sample}}$ represents the LLM generation process, where the model is prompted with the extracted data V, financial context C, original question, tool definitions $\mathcal{T}$ and is called to produce n independent program outputs by setting the sampling parameter $n = 15$.

4.4 Optimal Program Selection and Execution

The final stage involves selecting the optimal program p^* from the candidate set $\mathcal{P}_{\text{cand}}$ without supervised labels. We introduce a selection heuristic

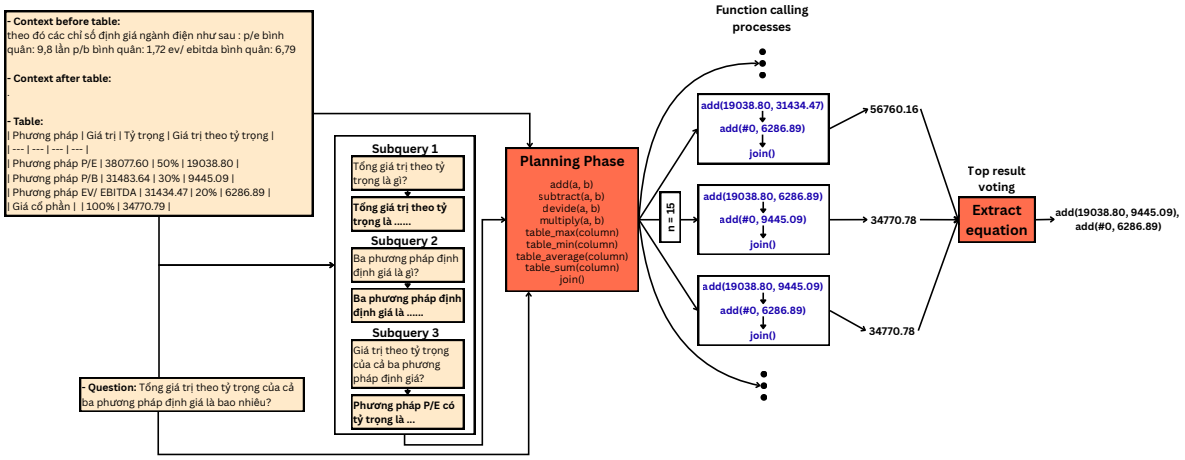

Figure 1: Overview of the graph-based AI agent pipeline for the VLSP 2025 Numerical Reasoning QA task, illustrating the four sequential nodes: Subquery Generator, Subquery Answerer, Plan and Scheduler, and Equation Extractor. Note: "Equation Extractor" in the diagram corresponds to the "Optimal Program Selection and Execution" stage described in the text.

based on majority voting over the generated program structures. First, the n candidate programs are canonicalized (e.g., arguments of commutative operators are sorted) and grouped to find the set of unique programs. The unique program that was generated most frequently is selected as the optimal one, p^*. This approach assumes that the reasoning path generated most consistently by the model is the most reliable. In case of a tie in frequency, the program with fewer steps (lower complexity) is chosen.

$$p^* = \arg\max_{p_i \in \mathcal{P}_{\text{cand}}} \text{Score}(p_i) \tag{4}$$

The $\text{Score}(p_i)$ function formalizes this by calculating the frequency of each unique, canonicalized program in $\mathcal{P}_{\text{cand}}$. The program with the highest frequency count receives the highest score, with complexity used as a tie-breaker. The final answer a^* is then obtained by executing this optimal program:

$$a^* = \text{Execute}(p^*) \tag{5}$$

The $\text{Execute}(p^*)$ function parses the program string, calls the corresponding tool functions with the specified arguments, and returns the final numerical result. The resulting pair (a^*, p^*) constitutes the final output of our system.

5 Experimental Setup

5.1 Implementation Details

Models For Subtask 1, we use the instruction-tuned **Qwen3-8B** model. For Subtask 2, we employ the larger **Qwen3-32B** model. This selection adheres to the competition's rules, which mandate a model size of $\leq$13B for Subtask 1 and allow for larger open-source models in Subtask 2. Both models are loaded in 'bfloat16' precision. To optimize inference throughput and manage memory efficiently, all experiments were conducted using the vLLM library (Kwon et al., 2023).

Configuration Our pipeline generates $k \in$ subqueries and uses an n-sampling parameter of $n = 15$ for program generation. Inference was performed using a sampling temperature of 0.6, with top_p and top_k set to 0.95 and 20, respectively. Experiments were run on two NVIDIA 80GB A100 GPUs.

5.2 Baseline Method

Our baseline uses the same foundation models and inference configuration but employs a direct, single-prompt approach. The model is prompted to generate the final reasoning program in one step,

allowing us to directly measure the performance gains from our structured workflow.

5.3 Evaluation Metrics

System performance is evaluated using two primary metrics: Execution Accuracy (EA) and Program Accuracy (PA). Let $(a_{\text{pred}}, p_{\text{pred}})$ be the predicted answer-program pair and $(a_{\text{gold}}, p_{\text{gold}})$ be the ground truth pair for a given question.

Execution Accuracy (EA) measures the percentage of predicted numerical answers that are an exact match to the gold answers. The metric is defined as:

$$\text{EA} = \frac{1}{N} \sum_{i=1}^{N} I(a_{\text{pred}}^{(i)} = a_{\text{gold}}^{(i)}) \quad (6)$$

where N is the total number of questions and $I(\cdot)$ is the indicator function, which is 1 if the condition is true and 0 otherwise.

Program Accuracy (PA) measures the percentage of predicted programs that are logically equivalent to the gold programs. Since semantically identical operations can have different surface forms (e.g., add(a, b) vs. add(b, a)), this metric evaluates structural equivalence after normalizing the program, such as by sorting arguments for commutative operators. It is defined as:

$$\text{PA} = \frac{1}{N} \sum_{i=1}^{N} I(\text{norm}(p_{\text{pred}}^{(i)}) \equiv \text{norm}(p_{\text{gold}}^{(i)})) \quad (7)$$

where $\text{norm}(\cdot)$ is the function that canonicalizes a program's structure and $\equiv$ denotes a string-level identity check. Table 1 provides illustrative examples for both metrics.

Metric	Gold	Prediction	Result
EA	2.338	2.338	Correct
EA	523.0	523.1	Incorrect
PA	divide(914, 391)	divide(914, 391)	Correct
PA	add(a, b)	add(b, a)	Correct
PA	subtract(a, b)	subtract(b, a)	Incorrect
PA	add(a, b)	multiply(a, b)	Incorrect

Table 1: Examples illustrating the evaluation criteria for Execution Accuracy (EA) and Program Accuracy (PA).

5.4 Experiment Results

We evaluate our agentic pipeline against the direct-prompting baseline and other top-performing systems in the VLSP 2025 Numerical Reasoning QA competition. Table 2 presents the head-to-head comparison against our baseline, while Table 3 situates our performance within the context of the final private test set leaderboard.

Method	Subtask	Public Test		Private Test	
		EA (%)	PA (%)	EA (%)	PA (%)
Baseline	Subtask 1 (8B)	41.05	32.60	-	-
MPR-Agent	Subtask 1 (8B)	**78.47**	**71.83**	**79.14**	**69.82**
Baseline	Subtask 2 (32B)	47.89	40.24	-	-
MPR-Agent	Subtask 2 (32B)	**81.29**	**75.25**	**84.00**	**74.07**

Table 2: Performance comparison against the direct-prompting baseline. Our agentic pipeline Multi-Path Reasoning Agent (MPR-Agent) achieves significant improvements across all metrics.

Subtask	Team (Method Description)	EA (%)	PA (%)	Avg Score (%)
Subtask 1	ngoquanghuy	**81.95**	75.00	78.48
	HUET	79.88	76.63	78.26
	Innovation-LLM (MPR-Agent)	**79.14**	**69.82**	**74.48**
	truong13012004	74.26	69.67	71.97
	vietld	68.49	61.83	65.16
	masterunited	56.80	54.14	55.47
Subtask 2	**Innovation-LLM (MPR-Agent)**	**84.00**	**74.07**	**79.04**
	HUET	79.88	76.63	78.26
	truong13012004	74.26	69.67	71.97

Table 3: Final private test set leaderboard results, ranked by Execution Accuracy (EA) to highlight performance in obtaining the correct final answer. Our team name is Innovation-LLM.

5.5 Analysis

The results unequivocally demonstrate the superiority of our structured agentic pipeline. As shown in Table 2, our method achieves a massive absolute improvement over the direct-prompting baseline, confirming that our structured approach is far more effective.

Table 3 situates our performance on the private test set, where we have ranked the results by Execution Accuracy (EA) to emphasize the model's primary strength: delivering the correct numerical answer. We also include an average score to provide a more holistic view of performance. In the highly competitive Subtask 2, our agent established its dominance by achieving the highest Execution Accuracy (84.00%) of any system. This result underscores our method's exceptional reliability in solving complex financial calculations, a crucial factor for real-world applications where correctness is paramount.

While the official competition was ranked by Program Accuracy (PA), our system's top-tier EA highlights a key insight into our design. The multi-path, consensus-based approach is optimized to find a valid and correct reasoning path, which may

not always syntactically match the single gold program. This phenomenon suggests that our agent prioritizes functional correctness over stylistic conformity. The high EA score, paired with a competitive PA, indicates that our system consistently identifies mathematically sound solutions, even if they differ in structure from the reference program. This flexibility is a feature, not a flaw, as it demonstrates robust problem-solving rather than rigid adherence to a single template (see Example 5.1 for a concrete case).

5.5.1 Ablation Study

To understand the contribution of each component in our pipeline, we conducted an ablation study on the public test sets for both Subtask 1 and Subtask 2 public test set. We evaluated three configurations: (1) our full MPR-Agent, (2) the pipeline without multi-path generation (Decomposition Only, n=1), and (3) the pipeline without initial question decomposition (Multi-Path Only). The results in Table 4 show that both decomposition and multi-path generation contribute to the final performance, with the multi-path providing the largest boost in Execution Accuracy.

Method Configuration	Model	EA (%)	PA (%)
MPR-Agent (Full Pipeline)	8B	78.47	71.83
Decomposition Only (n=1)	8B	72.84	62.58
Multi-Path Only (n=15)	8B	78.38	70.91
MPR-Agent (Full Pipeline)	32B	81.29	75.25
Decomposition Only (n=1)	32B	79.48	66.80
Multi-Path Only (n=15)	32B	80.91	74.65

Table 4: Ablation study of our pipeline components on the Subtask 1 & 2 public test set. Both decomposition and multi-path generation are crucial for performance.

5.5.2 Error Analysis and Heuristic Robustness

To investigate the failure modes of our consensus-based selection heuristic, we qualitatively analyzed a sample of failure cases from the public test set. Our findings indicate that failures primarily fall into two distinct categories:

1. **Systematic reasoning error:** In the majority of failures, the issue stemmed from the foundational reasoning capability of the base LLM itself. For certain complex or ambiguous questions, the model systematically failed to capture the correct logic, causing all, or nearly all, of the 15 generated programs to converge on the same flawed reasoning path. In these instances, our heuristic correctly selected the consensus program, but the consensus itself was fundamentally incorrect.

2. **Heuristic selection error:** In a smaller number of cases, the set of generated candidates did contain the correct program, but it was ultimately discarded. This failure occurs when one or more incorrect reasoning paths are generated more frequently than the single correct path, causing the correct program to be "outvoted". The heuristic, by design, selects the most frequent plan and thus fails. This represents an inherent limitation of the majority-voting mechanism, where a more common but flawed approach can overshadow a correct but less frequently generated one.

This analysis confirms that while the heuristic is effective at filtering out stochastic generation errors, its robustness is bounded by both the systematic reasoning capabilities of the LLM and the probabilistic nature of the selection process itself.

Example 5.1: Functional Correctness vs. Programmatic Equivalence

Question: Based on the results of the first quarter of 2019, what is the planned net profit after tax (LNST) for the entire year 2019?
Extracted Numbers: Net profit for Q1 2019 is 1,041, which completes 29.2% of the annual plan.
Gold Program: $\mathrm{divide}(1041,\ 0.292)$
Our Generated Program: $\mathrm{divide}(29.2,\ 100),\ \mathrm{divide}(1041,\ 0)$
Answer: Both programs execute to 3565.07.
Analysis: Our agent's generated program passes EA but fails PA. The gold standard directly uses the decimal value of the percentage (0.292). Our agent first explicitly converts the percentage to a decimal by dividing it by 100, and then uses that result for the final calculation. This demonstrates the agent's ability to find a functionally correct solution path by breaking the problem into more granular steps, even if it does not match the specific syntactic structure of the gold annotation.

For future work, aligning the model's output more closely with the canonical program struc-

tures of the training data could further improve PA scores. Fine-tuning the model on the competition dataset presents a promising path to harmonize our agent's powerful reasoning capabilities with the specific stylistic requirements of the benchmark. Nonetheless, our current inference-only method already demonstrates state-of-the-art performance in numerical accuracy, showcasing the immense power of advanced agentic workflows without the need for costly training overhead.

6 Limitations

While our proposed method demonstrates strong performance, we acknowledge several limitations. First, our optimal program selection relies on a consensus-based heuristic. As shown in our error analysis, this can fail when the underlying model systematically generates flawed reasoning paths, leading the majority to converge on an incorrect solution. A learned-critic or a more sophisticated validation mechanism could potentially mitigate this.

Second, the multi-path generation via n-sampling significantly increases computational overhead. Generating 15 candidate programs increases inference time and cost, which may not be practical for real-time applications. Our sensitivity analysis (see Appendix A) explores this trade-off, but it remains an inherent challenge of the approach.

Finally, our evaluation is confined to the VLSP 2025 dataset. The agent's performance on other financial reasoning benchmarks or in different domains has not been explored, and its effectiveness may vary depending on the complexity and structure of the data.

7 Conclusion

In this work, we introduced a structured, graph-based AI agent that effectively addresses the challenges of financial numerical reasoning. Our methodology, centered on question decomposition and multi-path plan generation via n-sampling, proved significantly more robust than a direct-prompting baseline. This was validated by our strong performance in the VLSP 2025 Numerical Reasoning QA competition, where our agent achieved the highest Execution Accuracy in Subtask 2 and secured top-3 EA and PA rankings in Subtask 1.

Our analysis revealed a powerful insight: our agent is exceptionally skilled at discovering functionally correct solutions, prioritizing the accuracy of the final answer over rigid adherence to a specific programmatic structure. This design choice resulted in our system achieving the highest Execution Accuracy in Subtask 2, underscoring its real-world applicability. The discrepancy between our leading EA and competitive PA is not a limitation, but rather a reflection of the system's robust, flexible reasoning. Future work can focus on aligning the agent's programmatic style with canonical formats through fine-tuning, which would bridge the gap between numerical correctness and structural equivalence, further advancing the development of accurate, transparent, and trustworthy AI for the financial domain.

References

Shanchuan Lin Rik Koncel-Kedziorski Yejin Choi Aida Amini, Saadia Gabriel and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1.

et al Chen, Zhiyu. 2021. Finqa: A dataset of numerical reasoning over financial data. *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Zhiyu Chen. 2022. Convfinqa: Exploring the chain of numerical reasoning in conversational finance question answering. *Preprint*, arXiv:2210.03849.

Pradeep Dasigi Gabriel Stanovsky-Sameer Singh Dheeru Dua, Yizhong Wang and Matt Gardner. 2019. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 1.

Morten Jerven. 2013. Poor numbers: how we are misled by african development statistics and what to do about it.

Yikyung Kim. 2024. Case-based reasoning approach for solving financial question answering. *Preprint*, arXiv:2405.13044.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody ho, Joseph E. Gonzalez, Ion Stoica, and Matei Zaharia. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles*.

Sanghoon Lee, Minchul Kim, and Jungmoon Kang. 2024. Enhancing financial question answering with a multi-agent framework incorporating a critic agent. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 1–8.

Donald MacKenzie. 2008. An engine, not a camera: How financial models shape markets.

Qwen Team. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Zichen Tang. 2025. Finmmr: Make financial numerical reasoning more multimodal, comprehensive, and challenging. *Preprint*, arXiv:2508.04625.

Zichen Tang, Haihong E, Ziyan Ma, Haoyang He, Jiacheng Liu, Zhongjun Yang, Zihua Rong, Rongjin Li, Kun Ji, Qing Huang, Xinyang Hu, Yang Liu, and Qianhe Zheng. 2025. Financereasoning: Benchmarking financial numerical reasoning more credible, comprehensive and challenging. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15721–15749, Vienna, Austria. Association for Computational Linguistics.

Jiaxin Zhang and Yashar Moshfeghi. 2022. Elastic: Numerical reasoning with adaptive symbolic compiler. *Advances in Neural Information Processing Systems*, 35:12647–12661.

Fengbin Zhu. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *Preprint*, arXiv:2105.07624.

Fengbin Zhu, Ziyang Liu, Fuli Feng, Chao Wang, Moxin Li, and Tat Seng Chua. 2024. Tat-llm: A specialized language model for discrete reasoning over financial tabular and textual data. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 310–318.

A Sensitivity Analysis of Hyperparameter n

We analyzed the sensitivity of our approach to the hyperparameter n, which controls the number of candidate programs generated. We evaluated Execution Accuracy (EA) and Program Accuracy (PA) on a subset of the validation set for $n \in \{1, 5, 10, 15, 20\}$. The results, shown in Figure A.1, reveal a nuanced trade-off. For EA, performance peaks at $n = 10$ and then slightly decreases. For PA, performance continues to improve, peaking at $n = 15$. Given that computational costs rise linearly with n, we chose $n = 15$ as it provided the best PA score while maintaining a high EA, striking a good balance between overall performance and inference efficiency.

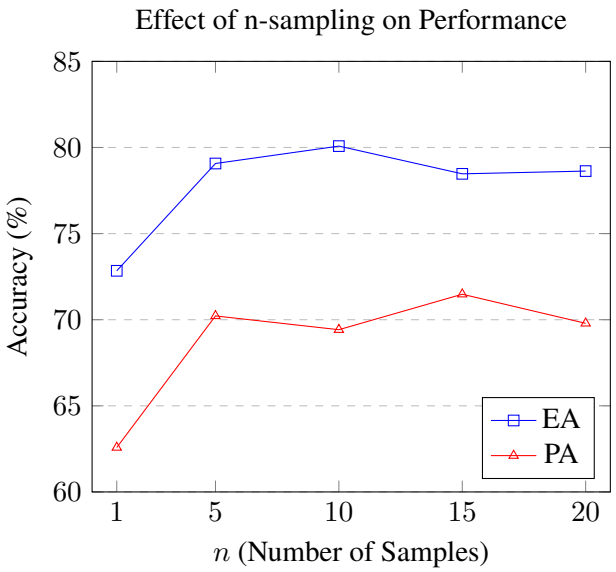

Figure A.1: Sensitivity analysis of the n-sampling parameter on the Subtask 1 public test set. Performance generally peaks between n=10 and n=15 before plateauing. Best Program Accuracy stop at n=15.

B Prompts

This appendix provides the full prompts used at each stage of our agentic pipeline. The prompt for the Subquery Generator, responsible for question decomposition, is shown in Appendix B.3. The prompt for the Subquery Answerer, which handles data extraction, is detailed in Appendix B.1. Finally, the planning stage uses a combination of a system prompt (Appendix B.5) and a comprehensive, multi-part user prompt (Appendices B.7, B.9, and B.11) to guide the generation of the reasoning program. English versions of these prompts are provided in Appendices B.4 through B.12.

Bạn là chuyên gia trong việc chia nhỏ câu hỏi của người dùng thành một chuỗi gồm 3 đến 5 câu hỏi con nhỏ hơn, dùng để trích xuất dữ liệu. Được đưa ra câu hỏi của người dùng và bối cảnh xung quanh (văn bản và bảng), nhiệm vụ của bạn là xác định các điểm dữ liệu thô cần thiết để trả lời câu hỏi chính.

Mục tiêu của bạn: Tạo danh sách các câu hỏi chỉ TRÍCH XUẤT giá trị dữ liệu thô hoặc văn bản hoặc phạm vi dữ liệu thô.

HƯỚNG DẪN QUAN TRỌNG:

- KHÔNG tạo ra các câu hỏi thực hiện so sánh (ví dụ, 'cái nào thấp nhất/cao nhất').
- KHÔNG tạo ra các câu hỏi thực hiện tính toán (ví dụ, 'tổng/trung bình/hiệu là bao nhiêu').
- KHÔNG tạo ra các câu hỏi yêu cầu câu trả lời cuối cùng.
- Công việc của bạn chỉ là hỏi về chính các điểm dữ liệu. Bước tiếp theo trong quy trình sẽ thực hiện tính toán thực tế.

Bây giờ, thực hiện nhiệm vụ này cho đầu vào sau đây.

Bối cảnh:
{context}

Câu hỏi gốc: {question}

Trả về các câu hỏi con dưới dạng đối tượng JSON với một khóa duy nhất "subqueries" chứa danh sách các chuỗi.

Figure B.1: Prompt for the Subquery Generator.

You are an expert at breaking down a user's question into a series of 3 to 5 smaller subqueries for data extraction. Given the user's question and surrounding context (text and tables), your task is to identify the raw data points needed to answer the main question.

Your goal: Generate a list of questions that ONLY EXTRACT raw data values or text or a raw data range.

IMPORTANT INSTRUCTIONS:

- DO NOT generate questions that perform comparisons (e.g., 'which is the lowest/highest').
- DO NOT generate questions that perform calculations (e.g., 'what is the total/average/difference').
- DO NOT generate questions that ask for the final answer.
- Your job is only to ask for the data points themselves. A later step in the pipeline will perform the actual calculations.

Now, perform this task for the following input.

Context:
{context}

Original Question: {question}

Return the subqueries as a JSON object with a single key "subqueries" containing a list of strings.

Figure B.2: English prompt for the Subquery Generator.

Bạn là trợ lý hữu ích. Nhiệm vụ của bạn là trả lời truy vấn con được đưa ra chỉ dựa trên bối cảnh được cung cấp.
Bối cảnh bao gồm văn bản trước bảng, chính bảng đó, và văn bản sau bảng. Luôn phản hồi bằng các câu hoàn chỉnh, nêu rõ câu trả lời cho câu hỏi hoặc tính toán.
Cung cấp câu trả lời ngắn gọn và trực tiếp cho truy vấn con.

Bối cảnh:
{context}

Truy vấn con:
{subquery}

Trả lời:

Figure B.3: Prompt for the Subquery Answerer.

You are a helpful assistant. Your task is to answer the given subquery based only on the provided context.
The context consists of the text preceding the table, the table itself, and the text following the table. Always respond with complete sentences, stating the answer to the question or calculation.
Provide a concise and direct answer to the subquery.

Context:
{context}

Subquery:
{subquery}

Answer:

Figure B.4: English prompt for the Subquery Answerer.

Bạn là một nhà lập kế hoạch tính toán số học, có nhiệm vụ tạo một chuỗi các lời gọi hàm để giải quyết truy vấn của người dùng với khả năng song song hóa tối đa, chỉ sử dụng các công cụ được cung cấp.

Figure B.5: System prompt for the Plan and Scheduler.

You are a numerical reasoning planner, tasked with generating a sequence of function calls to solve a user's query with maximum parallelization, using only the provided tools.

Figure B.6: English system prompt for the Plan and Scheduler.

Được đưa ra một truy vấn của người dùng, hãy tạo một chuỗi các lời gọi hàm để giải quyết với khả năng song song hóa tối đa chỉ sử dụng các công cụ sau:

1. add(a: str, b: str, context: Optional[list[str]]) -> float: Cộng hai đầu vào (chuỗi số hoặc biến).
2. subtract(a: str, b: str, context: Optional[list[str]]) -> float: Trừ đầu vào thứ hai từ đầu vào thứ nhất.
3. multiply(a: str, b: str, context: Optional[list[str]]) -> float: Nhân hai đầu vào.
4. divide(a: str, b: str, context: Optional[list[str]]) -> float: Chia đầu vào thứ nhất cho đầu vào thứ hai.
5. table_max(row_identifier: str) -> float: Trả về giá trị lớn nhất trong một hàng của bảng, được xác định bằng tên hàng chính xác.
6. table_min(row_identifier: str) -> float: Trả về giá trị nhỏ nhất trong một hàng của bảng, được xác định bằng tên hàng chính xác.
7. table_sum(row_identifier: str) -> float: Trả về tổng của một hàng trong bảng, được xác định bằng tên hàng chính xác.
8. table_average(row_identifier: str) -> float: Trả về trung bình của một hàng trong bảng, được xác định bằng tên hàng chính xác.
9. join(): Kết hợp kết quả cho phản hồi cuối cùng (phải được sử dụng làm bước cuối cùng).

Hướng dẫn:

- Bạn CHỈ ĐƯỢC sử dụng các công cụ được liệt kê ở trên. Không có công cụ nào khác tồn tại hoặc có thể được sử dụng.
- PHẦN TRĂM VS TỶ LỆ: Đối với bất kỳ truy vấn nào yêu cầu "phần trăm", "thay đổi phần trăm", hoặc "tỷ lệ", kế hoạch của bạn chỉ được tính tỷ lệ cuối cùng (ví dụ, 0.25). Bạn KHÔNG ĐƯỢC nhân tỷ lệ cuối cùng với '100' để chuyển đổi thành định dạng phần trăm (ví dụ, 25%). Hàm join chỉ cần trả về tỷ lệ đã tính cuối cùng.
- YÊU CẦU BẮT BUỘC: Đối với bất kỳ truy vấn nào yêu cầu kết quả số (ví dụ, hiệu, tổng, tỷ lệ, ROI, tác động tích lũy), bạn PHẢI tạo kế hoạch với các lời gọi công cụ số (ví dụ, add, subtract, multiply, divide) để tính toán hoặc xác minh kết quả. Việc sử dụng giá trị trực tiếp từ câu trả lời truy vấn con hoặc bảng mà không có lời gọi công cụ là HOÀN TOÀN BỊ CẤM, ngay cả khi câu trả lời có vẻ rõ ràng (ví dụ, '-23' trong câu trả lời truy vấn con hoặc bảng).
- QUAN TRỌNG - Thứ tự Phép trừ cho Thay đổi:
  - Đối với BẤT KỲ tính toán thay đổi nào (tăng, giảm, tăng trưởng, suy giảm, v.v.), LUÔN sử dụng: subtract(a='giá_trị_mới', b='giá_trị_cũ')
  - Có nghĩa là: giá trị hiện tại/cuối cùng/mới TRỪ giá trị trước đó/ban đầu/cũ.

Figure B.7: User prompt for the Plan and Scheduler (Part 1: Tools and Core Instructions).

Given a user query, generate a sequence of function calls to solve it with maximum parallelization using only the following tools:

1. add(a: str, b: str, context: Optional[list[str]]) -> float: Adds two inputs (numeric strings or variables).
2. subtract(a: str, b: str, context: Optional[list[str]]) -> float: Subtracts the second input from the first.
3. multiply(a: str, b: str, context: Optional[list[str]]) -> float: Multiplies two inputs.
4. divide(a: str, b: str, context: Optional[list[str]]) -> float: Divides the first input by the second.
5. table_max(row_identifier: str) -> float: Returns the maximum value in a table row, identified by its exact row name.
6. table_min(row_identifier: str) -> float: Returns the minimum value in a table row, identified by its exact row name.
7. table_sum(row_identifier: str) -> float: Returns the sum of a table row, identified by its exact row name.
8. table_average(row_identifier: str) -> float: Returns the average of a table row, identified by its exact row name.
9. join(): Combines results for the final response (must be used as the last step).

Instructions:

- You may ONLY use the tools listed above. No other tools exist or can be used.
- PERCENTAGE VS RATIO: For any query asking for a "percentage", "percent change", or "rate", your plan must only calculate the final ratio (e.g., 0.25). You MUST NOT multiply the final ratio by '100' to convert it to a percentage format (e.g., 25%). The join function should just return the final calculated ratio.
- MANDATORY REQUIREMENT: For any query that asks for a numerical result (e.g., difference, sum, ratio, ROI, cumulative impact), you MUST generate a plan with numerical tool calls (e.g., add, subtract, multiply, divide) to calculate or verify the result. Directly using a value from a subquery answer or table without a tool call is STRICTLY FORBIDDEN, even if the answer seems obvious (e.g., '-23' in a subquery answer or table).
- IMPORTANT - Subtraction Order for Change:
  - For ANY change calculation (increase, decrease, growth, decline, etc.), ALWAYS use: subtract(a='new_value', b='old_value')
  - Meaning: the current/final/new value MINUS the previous/initial/old value.

Figure B.8: English user prompt for the Plan and Scheduler (Part 1: Tools and Core Instructions).

Hướng dẫn (tiếp theo):

- Truy vấn Dựa trên Bảng:
  - Nếu truy vấn tham chiếu đến một bảng, trích xuất giá trị số từ bảng hoặc câu trả lời truy vấn con.
  - Đối với giá trị bảng có định dạng hỗn hợp (ví dụ, '$ -54 (54)'), sử dụng giá trị số (ví dụ, '-54').
  - Nếu bảng cung cấp giá trị cuối cùng (ví dụ, 'tổng ảnh hưởng lũy kế' = -23), xác minh nó bằng các lời gọi công cụ.
  - Sử dụng table_sum, table_max, v.v., chỉ cho các phép toán hàng.
- Đối với add, subtract, multiply, divide, đầu vào a và b PHẢI là chuỗi số (ví dụ, '3.14') hoặc biến (ví dụ, '$1') tham chiếu đến đầu ra của một hành động trước.
- Đối với các hàm bảng, row_identifier PHẢI là tên chuỗi chính xác của một hàng.
- Phân tích Đầu vào Số:
  - Đảm bảo tất cả đầu vào số được phân tích cùng đơn vị.
  - Đối với giá trị có dấu phần trăm (ví dụ, '48.8%'), loại bỏ dấu phần trăm và sử dụng giá trị số dưới dạng chuỗi (ví dụ, '48.8').
- Nếu "BỐI CẢNH BỔ SUNG TỪ TRUY VẤN CON" được cung cấp, hãy sử dụng các sự kiện và số liệu cụ thể từ bối cảnh đó để xây dựng kế hoạch của bạn.
- Biến ($x) CHỈ ĐƯỢC tham chiếu đến đầu ra của một hành động add, subtract, multiply, hoặc divide trước đó.
- Các biểu thức phức tạp (ví dụ, '3 * (4 + 5)') PHẢI được chia nhỏ thành các lời gọi công cụ riêng biệt.
- Kiểm tra và chuyển đổi các đơn vị để thống nhất trước khi tính toán.
- Đối với các truy vấn tài chính liên quan đến số tiền đô la (ví dụ, $47 triệu), sử dụng giá trị số dưới dạng chuỗi (ví dụ, '47').

Figure B.9: User prompt for the Plan and Scheduler (Part 2: Detailed Instructions).

Instructions (continued):

- Table-Based Queries:
  - If the query references a table, extract numeric values from the table or subquery answers.
  - For mixed-format table values (e.g., '$ -54 (54)'), use the numerical value (e.g., '-54').
  - If the table provides a final value (e.g., 'total cumulative effect' = -23), verify it with tool calls.
  - Use table_sum, table_max, etc., for row operations only.
- For add, subtract, multiply, divide, the inputs a and b MUST be numeric strings (e.g., '3.14') or variables (e.g., '$1') referencing the output of a prior action.
- For table functions, the row_identifier MUST be the exact string name of a row.
- Numeric Input Parsing:
  - Ensure all numeric inputs are parsed in the same units.
  - For values with a percentage sign (e.g., '48.8%'), strip the percent sign and use the numeric value as a string (e.g., '48.8').
- If "ADDITIONAL CONTEXT FROM SUBQUERY" is provided, use the specific facts and figures from that context to build your plan.
- Variables ($x) may ONLY refer to the output of a previous add, subtract, multiply, or divide action.
- Complex expressions (e.g., '3 * (4 + 5)') MUST be broken down into separate tool calls.
- Check and convert units for consistency before calculations.
- For financial queries involving dollar amounts (e.g., $47 million), use the numeric value as a string (e.g., '47').

Figure B.10: English user prompt for the Plan and Scheduler (Part 2: Detailed Instructions).

Hướng dẫn (cuối cùng):

- Mỗi hành động phải có một ID duy nhất, tăng nghiêm ngặt bắt đầu từ 1.
- Hành động join PHẢI là hành động cuối cùng và KHÔNG tạo ra đầu ra số.
- KHÔNG sử dụng $id của hành động join làm biến.
- Sau join, thêm <END_OF_PLAN>.
- Tối đa hóa khả năng song song hóa bằng cách cấu trúc các phép tính độc lập để thực hiện đồng thời.
- KHÔNG phát minh hoặc gọi các công cụ không tồn tại.

Ví dụ:
Truy vấn: Thay đổi giá trị hợp lý của các công cụ thị trường tài chính từ $47 triệu năm 2009 đến $21 triệu năm 2010
Kế hoạch: 1. subtract(a='21', b='47') 2. join() <END_OF_PLAN>

Truy vấn: Phần trăm thay đổi là bao nhiêu nếu doanh thu tăng từ $500 đến $600?
Kế hoạch: 1. subtract(a='600', b='500') 2. divide(a='$1', b='500') 3. join() <END_OF_PLAN>

Truy vấn: Tỷ suất lợi nhuận tích lũy của cổ phiếu Illumina Inc. trong bốn năm kết thúc vào năm 2003 là bao nhiêu, với giá trị 100.00 vào ngày 27 tháng 7 năm 2000 và 43.81 vào ngày 26 tháng 12 năm 2003?
Kế hoạch: 1. subtract(a='43.81', b='100.00') 2. divide(a='$1', b='100.00') 3. join() <END_OF_PLAN>

Truy vấn: Tổng của 'Lợi nhuận sau thuế (tỷ đồng)' cho tất cả các năm là bao nhiêu?
Bối cảnh: [Một bảng có hàng 'Lợi nhuận sau thuế (tỷ đồng)']
Kế hoạch: 1. table_sum(row_identifier='Lợi nhuận sau thuế (tỷ đồng)') 2. join() <END_OF_PLAN>

Chỉ trả lời với danh sách nhiệm vụ theo định dạng:
idx. tool(arg_name=args)
<END_OF_PLAN>

Figure B.11: User prompt for the Plan and Scheduler (Part 3: Final Rules and Examples).

Instructions (final):

- Each action must have a unique, strictly increasing ID starting from 1.
- The join action MUST be the final action and does NOT produce a numerical output.
- DO NOT use the $id of the join action as a variable.
- After the join, add <END_OF_PLAN>.
- Maximize parallelization by structuring independent calculations to run concurrently.
- DO NOT invent or call non-existent tools.

Examples:
Query: Change in the fair value of financial market instruments from $47 million in 2009 to $21 million in 2010
Plan: 1. subtract(a='21', b='47') 2. join() <END_OF_PLAN>

Query: What is the percentage change if revenue increases from $500 to $600?
Plan: 1. subtract(a='600', b='500') 2. divide(a='$1', b='500') 3. join() <END_OF_PLAN>

Query: What was the cumulative total return for Illumina Inc. stock for the four years ended 2003, given a value of 100.00 on July 27, 2000 and 43.81 on December 26, 2003?
Plan: 1. subtract(a='43.81', b='100.00') 2. divide(a='$1', b='100.00') 3. join() <END_OF_PLAN>

Query: What is the sum of 'Profit after tax (VND billion)' for all years?
Context: [A table with a 'Profit after tax (VND billion)' row]
Plan: 1. table_sum(row_identifier='Profit after tax (VND billion)') 2. join() <END_OF_PLAN>

Only respond with the task list in the format:
idx. tool(arg_name=args)
<END_OF_PLAN>

Figure B.12: English user prompt for the Plan and Scheduler (Part 3: Final Rules and Examples).